

Using Straggler Replication to Reduce Latency in Large-scale Parallel Computing (Extended Version)

Da Wang*

Two Sigma Investments, LLC.
New York, NY

dawang@alum.mit.edu

Gauri Joshi† Gregory Wornell

Signals, Information and Algorithms Laboratory
Research Laboratory of Electronics
Massachusetts Institute of Technology

{gauri,gww}@mit.edu

Abstract

Users expect fast and fluid response from today’s cloud infrastructure. Large-scale computing frameworks such as MapReduce divide jobs into many parallel tasks and execute them on different machines to enable faster processing. But the tasks on the slowest machines (straggling tasks) become the bottleneck in the completion of the job. One way to combat the variability in machine response time, is to add replicas of straggling tasks and wait for one copy to finish.

In this paper we analyze how task replication strategies can be used to reduce latency, and their impact on the cost of computing resources. We use extreme value theory to show that the tail of the execution time distribution is the key factor in characterizing the trade-off between latency and computing cost. From this trade-off we can determine which task replication strategies reduce latency, without a large increase in computing cost. We also propose a heuristic algorithm to search for the best replication strategies when it is difficult to fit a simple distribution to model the empirical behavior of task execution time, and use the proposed analysis techniques. Evaluation of the heuristic policies on Google Trace data shows a significant latency reduction compared to the replication strategy used in MapReduce.

1 Introduction

Applications such as Google search, Dropbox, Netflix need to perform enormous amounts of computing, or data processing on the cloud. Recently, cloud computing is also being offered as a service by Amazon S3, Microsoft Azure etc. where users can rent machines by the hour to run their computing jobs. The large-scale sharing of computing resources makes cloud computing flexible and scalable.

Cloud computing frameworks such as MapReduce [8] and Hadoop [18] employ massive parallelization to reduce latency. Large jobs are divided into hundreds of tasks that can be executed parallelly on different machines. An important class of parallel execution is “embarrassingly parallel” computation [22], which requires little or no effort in dividing the computation into independent parallel tasks. Several algorithms used in optimization and machine learning, for example the Alternating Direction Method of Multipliers (ADMM) [4], and Markov Chain Monte-Carlo (MCMC) [14], fall into this class and can be parallelized easily.

*Da Wang was affiliated with the Signals, Information and Algorithm Laboratory when this research was conducted.

†D. Wang and G. Joshi contributed equally to this work.

The execution time of a task on a machine is subject to stochastic variations due to co-hosting, virtualization and other hardware and network variations [11]. Thus, the key challenge in executing a job with a large number of tasks is the latency in waiting for the slowest tasks, or the “stragglers” to finish. As pointed out in [11, Table 1], the latency of executing many parallel tasks could be significantly larger (140 ms) than the median latency of a single task (1 ms).

In this work we analyze how replication of straggling tasks can be used to reduce latency. In particular, we develop a mathematical framework and provide insights into how task replication affects the trade-off between latency and computing cost.

1.1 Related prior work

The idea of replicating tasks in parallel computing has been recognized by system designers [5, 10], and first adopted at a large scale via the “backup tasks” in MapReduce [8]. A line of systems work [2, 3, 15, 23] further developed this idea to handle various performance variability issues in data centers.

While task replication has been studied in systems literature and also adopted in practice, there is not much work on careful mathematical analysis of replication strategies. In [21] replication strategies are analyzed, mainly for the single task case. In this paper we consider task replication in a computing job consisting of a large number of tasks, which corresponds closely to today’s large-scale cloud computing frameworks. We note that using replication or redundancy to reduce latency has also attracted attention in other contexts such as cloud storage and networking applications [12, 13, 17, 19].

1.2 Our contributions

To the best of our knowledge, we establish the first formal analysis of task replication in jobs consisting of a large number of tasks, for the system model and relevant performance measures proposed. We analyze the trade-off between the latency and the cost of computing resources, and provide insights into design of task replication strategies.

In particular, we study a class of scheduling policies called single-fork policies and characterize how latency and resource usage depend on three parameters: when we replicate tasks, how many replicas are launched, and whether the original replicas are killed or not. We show that the tail of the execution time distribution (heavy, light or exponential tail) is the key factor that determines the choice of the task replication policy. In particular for heavy tail distributions e.g. Pareto, we identify scenarios where the latency and computing cost can be reduced simultaneously. We also propose a heuristic algorithm to find the best single-fork policy when it is hard to use the proposed analysis techniques for the empirical distribution of task execution time.

1.3 Organization

The rest of the paper is organized as follows. In Section 2 we formulate the problem, define the latency and cost metrics and introduce notation used in the paper. In Section 3 we provide a summary of the analysis of single-fork task replication policies. Then in Section 4 we describe a heuristic algorithm to find a good scheduling policy for execution time distributions for which we cannot find an analytical distribution that fits well. Proofs of the analysis are given in Section 5. In Section 6 we conclude with a discussion of the implications and future perspectives. Finally, results from order statistics and extreme value theory used in this work are given in the Appendix.

2 Problem Formulation

We now describe the system model, and propose the performance metrics used to evaluate a task replication strategy.

2.1 Notation

First, we define some notation used in this paper. Lower-case letters (e.g., x) denote a particular value of the corresponding random variable, which is denoted in upper-case letters (e.g., X). We denote the cumulative distribution function (CDF) of X by $F_X(x)$. Its complement, the tail distribution is denoted by

$$\bar{F}_X(x) \triangleq 1 - F_X(x).$$

which may be more convenient to use than the c.d.f. sometimes. We denote the upper end point of F_X by

$$\omega(F_X) \triangleq \sup\{x : F_X(x) < 1\}. \quad (1)$$

For i.i.d. random variables X_1, X_2, \dots, X_n , we define $X_{j:n}$ as the j -th order statistic, i.e., the j -th smallest of the n random variables.

2.2 System Model

Consider a job consisting of n *parallel tasks*, where n is large. Analysis of real-world trace data shows that it is common for a job to contain hundreds or even thousands of tasks [16]. We assume the executing time of each task on a computing node is independent and identically distributed (i.i.d.) according to F_X , where F_X is the cumulative distribution function (CDF) of random variable X .

The distribution F_X accounts for the variability in the machine response due to various factors such as congestion, queuing, virtualization etc. We consider that there is an unlimited pool of machines such that each new task (or new replica) is assigned to a new machine. Hence, the execution time of a task can be assumed to be i.i.d. across machines.

Since we assume an unlimited pool of machines, the delay due to queueing of tasks at machines is small and can be subsumed as a constant additive term in the execution time X . Also, we do not consider the dependence of the task execution time on the size of the task itself. But it can be accounted for similarly by adding a constant delay (fixed across tasks of the same job) to the expected completion time of the job defined in Section 2.4.

2.3 Scheduling Policy

A scheduling policy or *scheduler* assigns tasks to different machines, possibly at different time instants. We assume that the scheduler receives instantaneous feedback notifying it when a machine finishes its assigned task. But there is *no intermediate feedback* indicating the status of processing of a task. When the scheduler receives notification that at least one replica of each of the n tasks has finished, it *kills all* the residual running replicas. It can also use the feedback to decide when to selectively launch or kill replicas in order to reduce the overall job completion time.

The times when the scheduler launches or kills replicas could be pre-determined (static policy) or be dependent on the feedback about the execution of other tasks in the job (dynamic policy). We focus our attention on a set of dynamic policies called single-fork policies, defined as follows.

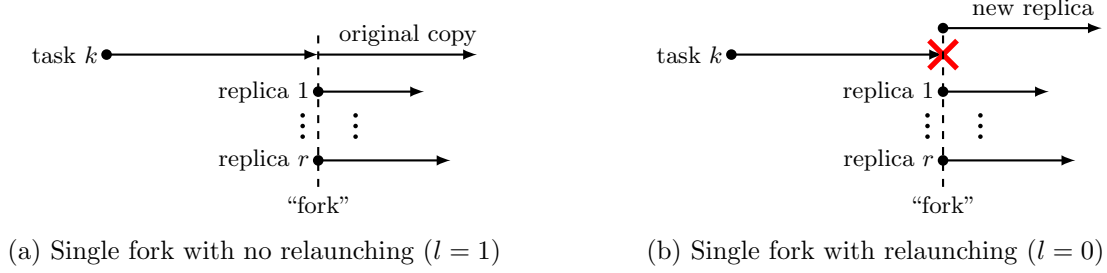


Figure 1: Illustration of single-fork policies with and without relaunching.

Definition 1 (Single-fork Scheduling policy). A single-fork scheduling policy $\pi_{\text{SF}}(p, r, l)$ launches a single replica of all n tasks at time 0. It waits until $(1 - p)n$ tasks finish and then for each of the pn straggling tasks, chooses one of the following two actions:

- **replicate without relaunching** ($l = 0$): launch r new replicas;
- **replicate with relaunching** ($l = 1$): kill the original copy and launch $r + 1$ new replicas.

When the earliest replica of a task is executed, all the other replicas are terminated.

We use l to denote the number of original replicas of each task remaining after the forking point. Hence $l = 0$ when the original replica is killed and restarted, and $l = 1$ otherwise. Note that in for both the relaunching ($l = 0$) and no relaunching ($l = 1$) cases there are a total of $r + 1$ replicas running after the forking point. The effect of r and l on the replication of straggling tasks is illustrated in Fig. 1.

For simplicity of notation we assume that p is such that pn is an integer. We note that $p = 0$ corresponds to running n tasks in parallel and waiting for all to finish, which is the baseline case without any replication or relaunching.

Remark 1 (Backup tasks in MapReduce). The idea of ‘backup’ tasks used in Google’s MapReduce [8], is a special case of the single-fork policy. Following our notation, it corresponds to $r = 1$ and $l = 1$. The value of p is tuned dynamically and hence not specified in [8].

Although we focus on single-fork policies in this paper, the analysis can be generalized to multi-fork policies, where new replicas of straggling tasks are launched at multiple times during the execution of the job. Forking multiple times can give a better latency-cost trade-off, but could be undesirable in practice due to additional delay and complexity in obtaining new and killing existing replicas.

2.4 Performance metrics

Our objective is to find the best single-fork policy $\pi_{\text{SF}}(p, r, l)$ for a given task execution time distribution F_X . We now define the performance metrics of latency and resource usage that are used to evaluate different scheduling policies.

Definition 2 (Expected Latency). The expected latency $\mathbb{E}[T]$ is the expected value of T , the time taken for at least one replica of each of the n tasks to finish. It can be expressed as,

$$\mathbb{E}[T] = \mathbb{E} \left[\max_{i \in \{1, 2, \dots, n\}} T_i \right], \quad (2)$$

where T_i is the minimum of the finish times of the replicas of task i , which depends on the scheduling policy used.

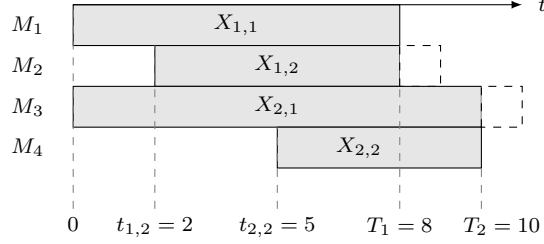


Figure 2: Illustration of T and C for a job with two tasks, and two replicas of each task. The latency $T = \max(8, 10) = 10$, and the computing cost is $C = (8 + 6 + 10 + 5)/2 = 14.5$.

Definition 3 (Expected Cost). *The expected computing cost $\mathbb{E}[C]$ is the sum of the running times of all machines, normalized by n , the number of tasks in the job. The running time is the time from when the task is launched on a machine, until it finishes, or is killed by the scheduler.*

For a user of a cloud computing service such as the Amazon Web Service (AWS), which charges the user by time and number of machines used, the money paid by the user to rent the machines is proportional to $\mathbb{E}[C]$.

Example 1. *Suppose the scheduler launches r replicas of each of the n tasks at times $t_{i,j}$ for $j = 1, 2, \dots, r$. Then the latency T_i of the i^{th} task is given by*

$$T_i = \min_{1 \leq j \leq r} (t_{i,j} + X_{i,j}), \quad (3)$$

where $X_{i,j}$ are i.i.d. draws from the execution time distribution F_X . The computing cost C can be expressed as

$$C \triangleq \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^r |T_i - t_{i,j}|^+, \quad (4)$$

where $|x|^+ = \max(0, x)$.

Fig. 2 illustrates the execution of a job with two tasks, and evaluation of the corresponding latency T and cost C . Given two tasks, we launch two replicas of task 1 $t_{1,1} = 0$ and $t_{1,2} = 2$, and two replicas of task 2 at $t_{2,1} = 0$ and $t_{2,2} = 5$. The task execution times are $X_{1,1} = 8$, $X_{1,2} = 7$, $X_{2,1} = 11$, and $X_{2,2} = 5$. Machine M_1 finishes the task first at time $t = 8$, $T_1 = 8$ and the second replica running on M_2 is terminated before it finishes executing. Similarly, machine M_4 finishes task 2 at time $T_2 = 10$, and the replica running on M_3 is terminated. Thus the latency of the job is $T = \max\{T_1, T_2\} = 10$. The cost is the sum of all running times normalized by n , i.e., $C = (8 + 6 + 10 + 5)/2 = 14.5$.

In this work we analyze the trade-off between $\mathbb{E}[T]$ and $\mathbb{E}[C]$ for the single-fork policy and provide insights into design of scheduling policies that achieve a good trade-off.

3 Single-fork policy analysis

In this section we evaluate the performance metrics $\mathbb{E}[T]$ and $\mathbb{E}[C]$ for a given single fork policy $\pi_{\text{SF}}(p, r, l)$. The main insight we get from this analysis is that the tail behavior (heavy, light or exponential) of F_X is the key factor in characterizing the latency-cost trade-off. We demonstrate this for two canonical distributions: Pareto and Shifted Exponential. All proofs are deferred to Section 5.

3.1 Performance characterization

For a job with a large number of tasks n , the expected latency and cost can be expressed in terms of the single-fork policy parameters p , r and l as given by Theorem 1 below.

Theorem 1 (Single-Fork Latency and Cost). *For a computing job with n tasks, and task execution time distribution F_X , the latency and cost metrics as $n \rightarrow \infty$ are*

$$\mathbb{E}[T] = F_X^{-1}(1-p) + \mathbb{E}[Y_{pn:pn}], \quad (5)$$

$$\mathbb{E}[C] = \int_0^{1-p} F_X^{-1}(h) dh + p F_X^{-1}(1-p) + (r+1)p \cdot \mathbb{E}[Y], \quad (6)$$

where F_Y is given by Lemma 2 below, and it is the distribution of the residual time after forking when the earliest replica of a straggling task finishes. The term $\mathbb{E}[Y_{pn:pn}]$ is the expected maximum of pn i.i.d. random variables drawn from F_Y . Its behavior for $n \rightarrow \infty$ is given by Lemma 3 below.

Lemma 2 (Residual Straggler Execution Time). *As $n \rightarrow \infty$, the tail distribution \bar{F}_Y of the residual execution time (after the forking point) of each of the pn straggling tasks is*

$$\bar{F}_Y(y) = \begin{cases} \bar{F}_X(y)^{r+1} & \text{if } l = 0, \\ \frac{1}{p} \bar{F}_X(y)^r \bar{F}_X(y + F_X^{-1}(1-p)) & \text{if } l = 1. \end{cases} \quad (7)$$

Lemma 3. *As $n \rightarrow \infty$, the asymptotic behavior of $\mathbb{E}[Y_{pn:pn}]$ is given by*

1. If $F_Y \in \text{DA}(\Lambda)$,

$$\mathbb{E}[Y_{pn:pn}] = \tilde{a}_{pn} \gamma_{\text{EM}} + \tilde{b}_{pn},$$

2. If $F_Y \in \text{DA}(\Phi_{(r+1)\xi})$,

$$\mathbb{E}[Y_{pn:pn}] = \tilde{a}_{pn} \Gamma(1 - 1/[(r+1)\xi]),$$

3. If $F_Y \in \text{DA}(\Psi_{((1-l)r+1)\xi})$,

$$\mathbb{E}[Y_{pn:pn}] = \omega(F_Y) - \tilde{a}_{pn} \Gamma(1 + 1/[(1-l)r+1)\xi]),$$

where $\text{DA}(\cdot)$ is the domain of attraction of F_Y which can be determined using Lemma 9 and Theorem 12. The terms \tilde{a}_{pn} and \tilde{b}_{pn} are the normalizing constants of F_Y as given by Theorem 11, γ_{EM} is the Euler-Mascheroni constant, and $\Gamma(\cdot)$ is the Gamma function, i.e.,

$$\Gamma(t) \triangleq \int_0^\infty x^{t-1} e^{-x} dx. \quad (8)$$

The domain of attraction of a distribution depends on its tail behavior (exponential, heavy or light). For example, exponentially decaying distributions belong to $\text{DA}(\Lambda)$ while heavy tailed distributions belong to $\text{DA}(\Phi_\xi)$.

We now give a sketch of the proof of Theorem 1. A detailed proof can be found in Section 5.

Proof sketch of Theorem 1 . The expected latency of a single fork policy $\pi_{\text{SF}}(p, r, l; n)$ can be decomposed into two parts:

$$\mathbb{E}[T] = \mathbb{E}[T^{(1)}] + \mathbb{E}[T^{(2)}], \quad (9)$$

where, $T^{(1)}$ is the time to execute the first $(1-p)n$ tasks and $T^{(2)}$ is the time to execute the rest of the pn tasks with replication. We evaluate each of these parts separately.

It is straightforward to see that $T^{(1)}$ is the $((1-p)n)^{\text{th}}$ order statistic of n i.i.d. random variables with distribution F_X . Thus its expected value is

$$\mathbb{E}[T^{(1)}] = \mathbb{E}[X_{(1-p)n:n}], \quad (10)$$

$$\approx F_X^{-1}(1-p) \quad \text{for large } n, \quad (11)$$

where (11) follows from the Central Value Theorem (Theorem 10) which states that the $((1-p)n)^{\text{th}}$ order statistic concentrates sharply around $F_X^{-1}(1-p)$ as $n \rightarrow \infty$.

After $n(1-p)$ tasks finish, the scheduler adds redundancy by launching replicas of the straggling tasks and waits for one replica to finish. We denote the residual execution time distribution of the straggling tasks by F_Y . It depends of F_X and the parameters r, p and l of the scheduling policy as given by Lemma 2. For example, for $r = 2$ and $l = 0$, the tail of distribution $\bar{F}_Y = \bar{F}_X^2$, which is the minimum of two i.i.d. random variables with distribution F_X . The proof of Lemma 2 is given in Section 5.

The second part of the latency, $T^{(2)}$ is the maximum of the times until each of the pn straggling tasks finish. Thus, its expected value $\mathbb{E}[T^{(2)}] = \mathbb{E}[Y_{pn:pn}]$. The behavior of the maximum order statistic of a large number of random variables is given by the Extreme Value Theorem Theorem 11. We can use it to show that $\mathbb{E}[T^{(2)}]$ is given by Lemma 3.

Thus we can evaluate the expected latency $\mathbb{E}[T] = \mathbb{E}[T^{(1)}] + \mathbb{E}[T^{(2)}]$. Similarly, the expected cost $\mathbb{E}[C]$ can be evaluated by decomposing it into two parts: before and after the replication of straggling tasks. The details can be found in the proof in Section 5. \square

3.2 Examples of the Effect of Tail Behavior

We now demonstrate how the the tail of the distribution F_X is a major factor in determining the trade-off between $\mathbb{E}[T]$ and $\mathbb{E}[C]$, and hence the choice of the best single-fork policy. We consider two canonical execution time distributions, the Pareto distribution (heavy tailed) and the Shifted Exponential distribution (exponential tail) and evaluate the latency-cost trade-off in Theorem 1 for them. One key insight from this analysis is that in certain regimes, it is possible to reduce latency while simultaneously reducing cost.

3.2.1 Pareto execution time

The cumulative distribution function of the Pareto distribution $\text{Pareto}(\alpha, x_m)$ is

$$F(x; \alpha, x_m) \triangleq \begin{cases} 1 - \left(\frac{x_m}{x}\right)^\alpha & x \geq x_m, \\ 0 & x < x_m. \end{cases} \quad (12)$$

Pareto distribution is a heavy-tail distribution, with a polynomially decaying tail. It has been observed to fit task execution time distributions in data centers [11, 16].

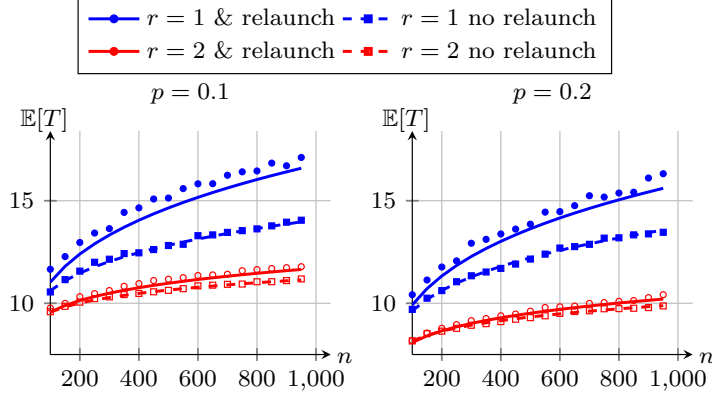


Figure 3: Comparison of the expected latency $\mathbb{E}[T]$ obtained from simulation (points) and analytical calculations (lines) for the Pareto distribution Pareto(2, 2).

Theorem 4. For a computing job with n tasks, if the execution time distribution of each task is Pareto(α, x_m), then as $n \rightarrow \infty$, the latency and cost metrics are

$$\mathbb{E}[T] = x_m p^{-1/\alpha} + \Gamma \left(1 - \frac{1}{(r+1)\alpha} \right) \tilde{a}_{pn}, \quad (13)$$

$$\mathbb{E}[C] = x_m \frac{\alpha}{\alpha - 1} - x_m \frac{p^{1-1/\alpha}}{\alpha - 1} + (r+1)p\mathbb{E}[Y]. \quad (14)$$

The values of \tilde{a}_{pn} and $\mathbb{E}[Y]$ depend on the relaunching parameter l , and are given as follows.

Case 1: Relaunching ($l = 0$)

$$\tilde{a}_{pn} = (pn)^{\frac{1}{(r+1)\alpha}} x_m, \quad (15)$$

$$\mathbb{E}[Y] = \frac{(r+1)\alpha}{(r+1)\alpha - 1} x_m. \quad (16)$$

Case 2: No Relaunching ($l = 1$)

The term \tilde{a}_{pn} is the solution to

$$n^{1/\alpha} x_m^{r+1} = x_m p^{-1/\alpha} \tilde{a}_{pn}^r + \tilde{a}_{pn}^{r+1}. \quad (17)$$

and $\mathbb{E}[Y]$ is evaluated numerically as discussed in the proof.

The proof is given in Section 5. From this theorem we can infer that the latency $\mathbb{E}[T]$ grows polynomially $o(n^{1/\alpha(r+1)})$ with n . This can be seen directly from (15) for the case of relaunching ($l = 0$). For the no relaunching case we know that \tilde{a}_{pn} grows with n . Thus for large enough n , it should be greater than $x_m p^{-1/\alpha}$. Hence from (17),

$$n^{1/\alpha} x_m^{r+1} \leq 2\tilde{a}_{pn}^{r+1}. \quad (18)$$

The $o(n^{1/\alpha(r+1)})$ growth follows from this.

Fig. 3 compares the latency obtained from simulation and analytical calculations for Pareto distribution, indicating latency obtained from analytical calculation is very close to the actual performance for $n \geq 100$, especially for the case with relaunching ($l = 0$). In Fig. 4 we plot the expected latency and cost as p varies, for different values of r and l . The black dot is the baseline

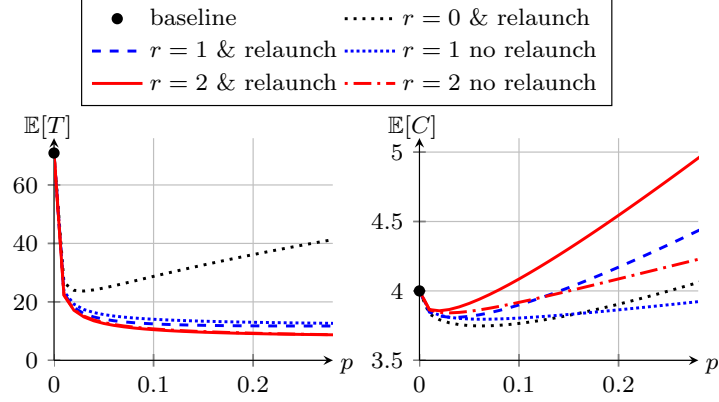


Figure 4: Expected latency and cloud user cost for a Pareto execution time distribution $\text{Pareto}(2, 2)$, given $n = 400$.

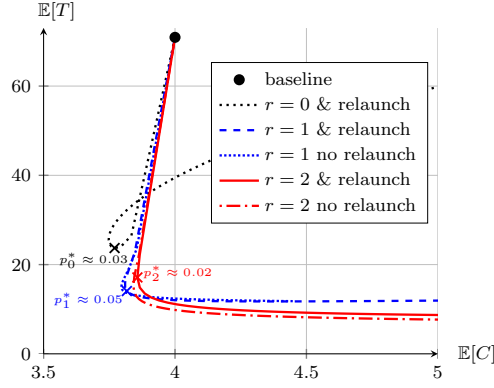


Figure 5: Expected latency $\mathbb{E}[T]$ versus the expected cost $\mathbb{E}[C]$ for $\text{Pareto}(2, 2)$ and $n = 400$, by varying p along each curve in the range of $[0, 1]$. For small p , we can reduce both latency and cost simultaneously.

case ($p = 0$), where no replication is used and we simply wait for the original copies of all n tasks to finish. The baseline case is also equivalent to the policies with $r = 0, l = 1$ and any p .

In Fig. 4 we observe that a small amount of replication (small p and r) can reduce latency significantly in comparison with the baseline case. But as p increases further, the latency may increase (as observed for $r = 0$) because of the second term in (2). For a given r , relaunching leads to lower latency when p satisfies the condition in Lemma 5 below.

Lemma 5. *For given r , relaunching ($l = 0$) gives lower latency $\mathbb{E}[T]$ than no relaunching ($l = 1$) when p satisfies,*

$$p^{1/\alpha} + (np)^{-1/(r+1)\alpha} \leq 1, \quad (19)$$

where α is the shape parameter of the Pareto distribution, as given in (12).

Intuition suggests that replicating earlier (larger p) and more (higher r) will increase the cost $\mathbb{E}[C]$. But Fig. 4 shows that this is not necessarily true. Since we kill all the machines running a task when one of its replicas finish, there is in fact a saving in the computing cost! However this benefit diminishes as p and r increase above a certain threshold.

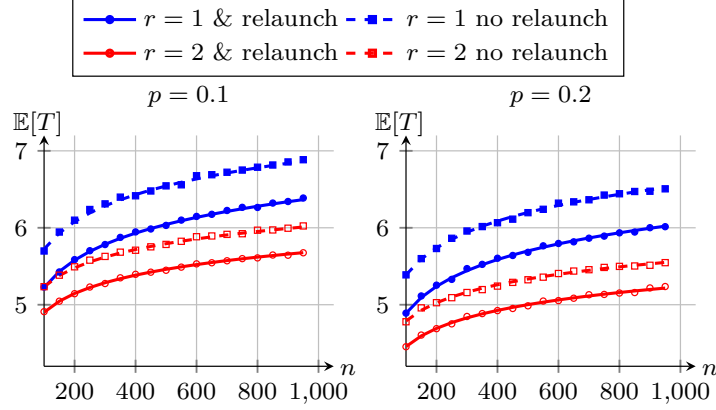


Figure 6: Comparison of the expected latency $\mathbb{E}[T]$ obtained from simulation (points) and analytical calculations (lines) for the Shifted Exponential distribution $\text{SExp}(1, 1)$.

Fig. 5 shows the latency versus the computing cost for different values of r and l , with p varying along each curve. Depending upon the latency requirement and limit on the cost, one can choose an appropriate operating point on this trade-off. This plot demonstrates the non-intuitive phenomenon that it is possible to reduce latency (from 70 to about 15 for $r = 1$ and $r = 2$ cases) with negative cost! In Lemma 6 we identify the values of p for the relaunching ($l = 0$) case when the corresponding single-fork policy is sub-optimal in both $\mathbb{E}[T]$ and $\mathbb{E}[C]$.

Lemma 6. *For a given r and relaunching ($l = 0$), the range of p for which the single-fork policy is sub-optimal is given by*

$$\left(\Gamma \left(1 - \frac{1}{(r+1)\alpha} \right) n^{1/(r+1)\alpha} - (r+1)p^{-(r+2)/(r+1)\alpha} \right) \cdot \left(\frac{(r+1)^2\alpha}{(r+1)\alpha - 1} - \frac{p^{-1/\alpha}}{\alpha} \right) = 0, \quad (20)$$

where a policy $\pi_{\text{SF}}(p, r, l)$ is said to be sub-optimal if there exists another policy $\pi_{\text{SF}}(p', r, l)$ which gives lower $\mathbb{E}[T]$ and $\mathbb{E}[C]$ than $\pi_{\text{SF}}(p, r, l)$.

For example for the $r = 1$ and relaunch ($l = 0$) case, we can solve (20) to show that all policies with $p < p_1^* \approx 0.05$ are sub-optimal, where p_1^* is marked in Fig. 5. Similarly, for cases $r = 0$ and $r = 2$, the sub-optimal ranges $[0, p_0^*]$ and $[0, p_2^*]$ are shown respectively in Fig. 5.

We conjecture that the convex hull of the curves for different r and l gives the optimal latency-cost trade-off. Points on the hull, which lie between some two curves can be achieved by time-sharing between the corresponding two policies.

3.2.2 Shifted Exponential execution time

We now analyze the latency-cost trade-off when the task execution time distribution F_X is a Shifted Exponential Distribution $\text{SExp}(\Delta, \lambda)$. Unlike the Pareto distribution which is heavy-tailed, shifted exponential has an exponentially decaying tail. Its cumulative distribution function is given by

$$F(x) = \begin{cases} 1 - e^{-\lambda(x-\Delta)} & \text{for } x \geq \Delta, \\ 0 & \text{otherwise.} \end{cases} \quad (21)$$

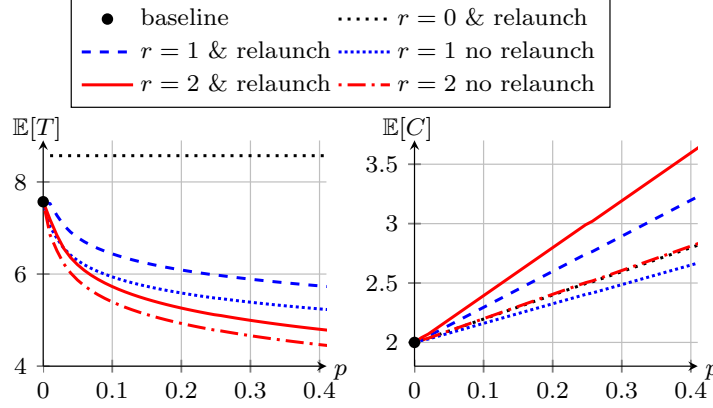


Figure 7: Expected latency and cost for a Shifted Exponential execution time distribution $\text{SExp}(1, 1)$, given $n = 400$.

The special case $\Delta = 0$ corresponds to the pure exponential distribution, which is popular in queueing theory and scheduling due to its memoryless property. But it may not be suitable for modeling the execution time of a task, as a task seldom finishes instantaneously, and usually it is lower bounded by a constant delay due to machine start-up or task initialization. Hence we add the constant delay Δ and model the execution time using the $\text{SExp}(\Delta, \lambda)$.

Theorem 7. *For a computing job with n tasks, if the execution time distribution of each task is $\text{SExp}(\Delta, \lambda)$, then as $n \rightarrow \infty$, the latency and cost metrics are*

$$\mathbb{E}[T] = \frac{2r + l}{r + l} \Delta + \frac{1}{(r + 1)\lambda} (\ln n - r \ln p + \gamma_{\text{EM}}), \quad (22)$$

$$\mathbb{E}[C] = \begin{cases} \Delta + \frac{1}{\lambda} + p \left[\Delta + r \frac{(1 - e^{-\lambda\Delta})}{\lambda} \right] & l = 1, \\ \Delta + \frac{1}{\lambda} + p(r + 2)\Delta & l = 0. \end{cases} \quad (23)$$

Similar to Fig. 3, Fig. 6 compares the latency obtained from simulation and analytical calculations for the Shifted Exponential distribution, which again demonstrates the effectiveness of the asymptotic theory.

We can draw the following observations from Theorem 7. Given r and l , replicating earlier (larger p) gives an $\Theta(\ln p)$ decrease in latency, and a linear increase the cost. This is also illustrated in Fig. 7 for execution time distribution $\text{SExp}(1, 1)$ and $n = 400$. Fig. 8 illustrates the latency-cost trade-off. Unlike Fig. 5 there is no range of p for which both latency and cost decrease (or increase) simultaneously.

For the special case of $\Delta = 0$ by Theorem 7, the cost $\mathbb{E}[C] = \frac{1}{\lambda}$, which is independent of p and r . But latency always reduces with r and p . This suggests that we can achieve arbitrarily low latency without any increase in cost. Since this is not observed in practice, we can conclude that the pure exponential distribution is not a useful model for the task execution time.

For a given p and r , relaunching always gives larger latency than no relaunching. But the cost may increase or decrease depending on the values of λ and Δ . Lemma 8 below gives the set of the parameters for which latency and cost are strictly larger with relaunching.

Lemma 8. *If $\Delta > \beta^*/\lambda$, where $\beta^* > 0$ is the solution to*

$$\beta r + \beta - r + r e^{-\beta} = 0,$$

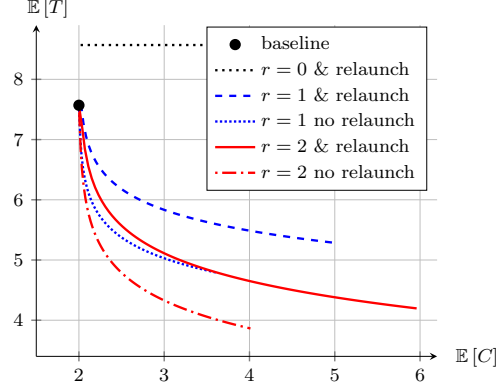


Figure 8: The trade-off between expected latency $\mathbb{E}[T]$ and expected cost $\mathbb{E}[C]$ for SExp(1, 1) and $n = 400$, by varying p in the range of $[0.05, 0.95]$.

Algorithm 1 Estimate Latency and Cost Metrics

Find empirical CDF $F_X(x)$ from the latency traces
Find CDF $F_Y(y)$ using Lemma 2
for $i = 1, 2, \dots, m$ **do**
 Draw n samples from F_X
 $\tilde{T}_1^{(i)} \leftarrow$ the $n(1-p)^{th}$ smallest sample
 $\tilde{C}_1^{(i)} \leftarrow$ the mean of the smallest $n(1-p)$ samples
 Draw np samples from F_Y
 $Y_{max}^{(i)} \leftarrow$ the maximum of the samples
 $Y_{avg}^{(i)} \leftarrow$ the mean of the samples
 $\tilde{T}^{(i)} \leftarrow \tilde{T}_1^{(i)} + Y_{max}^{(i)}$
 $\tilde{C}^{(i)} \leftarrow \tilde{C}_1^{(i)} + p\tilde{T}_1^{(i)} + (r+1)Y_{avg}^{(i)}$
end for
 $\tilde{T} \leftarrow$ mean of $\tilde{T}^{(i)}$ for $i = 1, 2, \dots, m$
 $\tilde{C} \leftarrow$ mean of $\tilde{C}^{(i)}$ for $i = 1, 2, \dots, m$

then relaunching leads to strictly larger latency and cloud computing cost than no relaunching. In particular, $\beta^* < 1$, hence if $x_m \geq 1/\lambda$, then no relaunching achieves better trade-off between latency and cost than relaunching.

4 Heuristic Algorithm

In certain practical systems may be difficult to fit a well-known distribution such as Pareto or shifted exponential to the empirical behavior of the task execution time. This makes the analysis of the latency-cost trade-off using the framework presented in Section 3 hard. In this section we present algorithms that use traces of task execution times to estimate the latency and cost metrics, and use these estimates to search for the best single-fork policy $\pi_{SF}(p, r, l)$. We also evaluate the algorithms on Google trace data [1].

4.1 Estimation of Latency and Cost Metrics

In Algorithm 1 we propose an algorithm to estimate $\mathbb{E}[T]$ and $\mathbb{E}[C]$ based on random sampling of distributions F_X and F_Y . The estimates are denoted by \tilde{T} and \tilde{C} respectively.

We first construct empirical CDFs \hat{F}_X and \hat{F}_Y from experimental traces of start and finish times of tasks on a large cluster of machines. The traces need to be sufficient to capture the tail behavior of the distribution, which plays a key role in characterizing the latency-cost trade-off, as seen in Section 3. To estimate the terms of $\mathbb{E}[T]$ and $\mathbb{E}[C]$ in (5) and (6) involving F_X , we draw n i.i.d. samples from \hat{F}_X and find the maximum and mean of the smallest $n(1-p)$ samples. Similarly, to estimate the terms in (5) and (6) involving Y , we draw pn samples from \hat{F}_Y and find the maximum and mean of the samples. The above steps are repeated m times and \tilde{T} and \tilde{C} are set to means of the corresponding estimates from each step.

While it is possible to estimate \tilde{T} and \tilde{C} directly from traces of the execution time, we take the two-step approach of first finding the empirical CDFs \hat{F}_X and \hat{F}_Y and then estimating \tilde{T} and \tilde{C} . This is because unlike samples of the task execution time X , the samples of the residual straggler execution time Y (Lemma 2) cannot be directly obtained from the traces of task execution time. Also, this two-step approach allows system designers to improve each of two estimates separately. For example, the CDFs \hat{F}_X and \hat{F}_Y can be smoothed using bootstrapping methods [9].

By the Central Limit Theorem we can show that the standard deviation of the error in estimating $\mathbb{E}[C]$, and the first term in $\mathbb{E}[T]$ converges to zero as $O(1/\sqrt{mn})$, where m is the number of times the sampling procedure is repeated. But in general, the maximum order statistic term in $\mathbb{E}[T]$ converges to zero as $O(1/\sqrt{m})$, which is slower. Thus, the estimate of \tilde{C} is more robust than that of \tilde{T} .

4.2 Heuristic Search for the Best Policy

We now present a heuristic algorithm that uses traces of execution time to search for the best single-fork policy. This policy can be used to perform task replication in future jobs that have similar task execution time statistics. The best single-fork policy is said to be the policy that minimizes the objective function J defined as,

$$\begin{aligned} J &\triangleq \mathbb{E}[T] + \mu \mathbb{E}[C], \\ &\approx \tilde{T} + \mu \tilde{C}. \end{aligned} \tag{24}$$

The parameter μ is the priority given to the minimizing the cost. Since we know only estimates \tilde{T} and \tilde{C} of the expected latency and cost from Algorithm 1, we use the estimated objective function in (24).

Algorithm 2 gives the pseudo-code of the search for the best single-fork parameters p , r and l . For a given p , we first find the optimal r and l , and then perform gradient descent on p . This is repeated for k iterations. To optimize r and l we keep increasing r by 1 until the objective function decreases. For each r , we set l to the value (0 or 1) which gives a smaller J . The terms $\tilde{T}(\pi)$ and $\tilde{C}(\pi)$ are the latency and cost estimates for the policy $\pi = \pi_{\text{SF}}(p, r, l)$ found using Algorithm 1.

The policy found by Algorithm 2 may not be the true optimal single-fork policy due to the following error factors:

1. Error in the estimates of $\mathbb{E}[T]$ and $\mathbb{E}[C]$ from Algorithm 1.
2. The gradient descent in p could be slow and may not converge to the optimum in k iterations. Also, note that $\mathbb{E}[T]$ and $\mathbb{E}[C]$ in (5) and (6) are convex in r , but not in p and l . Thus, the algorithm is not guaranteed to converge to the optimal single-fork policy.

Algorithm 2 Heuristic to Find Best Single-Fork Policy

```
Initialize  $p = 0, r^* = 0, l^* = 0$ 
for  $i = 1, 2, \dots k$  do
  % For given  $p$ , optimize  $l$  and  $r$ 
  while 1 do
     $\pi \leftarrow \pi_{\text{SF}}(p, r^*, l^*)$ 
     $\pi'_0 \leftarrow \pi_{\text{SF}}(p, r^* + 1, 0)$ 
     $\pi'_1 \leftarrow \pi_{\text{SF}}(p, r^* + 1, 1)$ 
    if  $\tilde{T}(\pi'_1) + \mu\tilde{C}(\pi'_1) < \tilde{T}(\pi'_0) + \mu\tilde{C}(\pi'_0)$  then
       $\pi' \leftarrow \pi'_1$ 
    else
       $\pi' \leftarrow \pi'_0$ 
    end if
     $\Delta_J \leftarrow \tilde{T}(\pi') - \tilde{T}(\pi) + \mu(\tilde{C}(\pi') - \tilde{C}(\pi))$ 
    if  $\Delta_J < 0$  then
       $\pi \leftarrow \pi'$ 
    else
      break
    end if
  end while

  % Gradient Descent on  $p$ 
   $\pi_{\text{SF}}(p, r^*, l^*) \leftarrow \pi$ 
   $\pi' \leftarrow \pi_{\text{SF}}(p + \Delta_p, r^*, l)$ 
   $\Delta_J \leftarrow \tilde{T}(\pi') - \tilde{T}(\pi) + \mu(\tilde{C}(\pi') - \tilde{C}(\pi))$ 
   $p \leftarrow p - \Delta_p \Delta_J$ 
end for
```

3. The task execution time statistics of future jobs may not match the empirical CDF \hat{F}_X that was used to find the best policy in Algorithm 2.

4.3 Demonstration using Google Traces

We now demonstrate the results of running Algorithm 2 on Google Trace data [1]. We use the traces to estimate the distribution F_X and then run the heuristic algorithm on it to find the best single-fork policy.

The Google Trace data gives timestamps of events such as SCHEDULE, EVICT, FINISH, FAIL, KILL etc. for each of the tasks of computing jobs that are run on Google's machine clusters. We consider the difference between the SCHEDULE and FINISH timestamps as the task execution time, and construct the empirical distribution \hat{F}_X . Note that because we consider only the SCHEDULE and FINISH task, we are not accounting for the computing resources consumed due to failure or eviction of tasks.

We consider two large Google cluster jobs with $n = 1017$ and $n = 488$ tasks respectively. Their normalized histograms are shown in Fig. 9 and Fig. 10. Fig. 9 shows heavy-tail behavior of task execution time, whereas Fig. 10 has bimodal behavior with a very small percentage of tasks finishing in around 1400 seconds.

Then we run Algorithm 2 on the empirical \hat{F}_X found using the histograms. We use $\Delta_p = 0.002$,

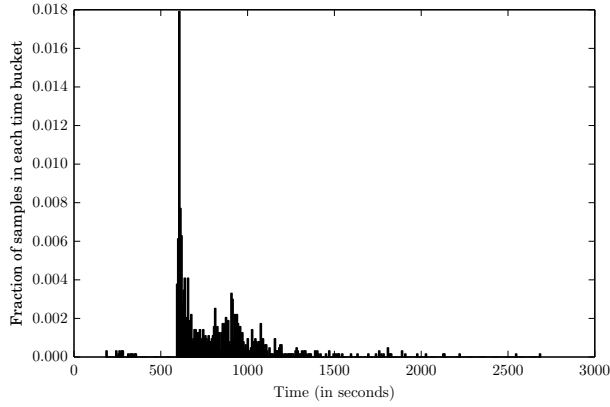


Figure 9: Normalized histogram of the task execution times for a Google cluster job with $n = 1017$ tasks.

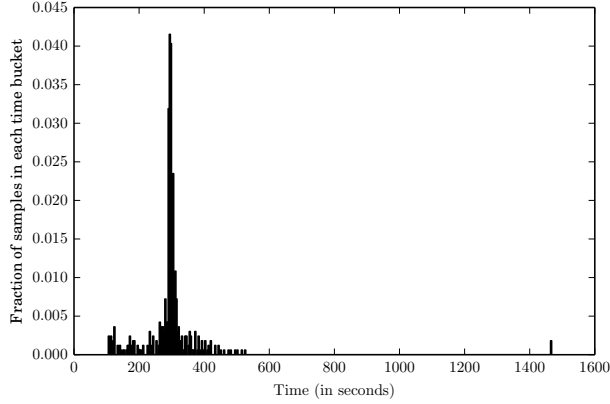


Figure 10: Normalized histogram of the task execution times for a Google cluster job with $n = 488$ tasks.

$m = 500$ and $k = 25$. The latency-cost trade-offs of the heuristic policies found by the algorithm are shown in Fig. 11 and Fig. 12. We also plot the estimated latency-cost trade-off for $r = 1$ and $l = 1$ as p varies from 0 to 1. These are the parameters of the back-up tasks option in MapReduce as described in Remark 1.

We observe in Fig. 11 that the heuristic algorithm finds policies with $r > 1$ that give lower latency for the same cost, than the policies with $r = 1$ and $l = 1$. We also observe that the latency reduction is more when μ is smaller, that is, the priority given to minimizing the cost is lower. But in Fig. 12, the reduction in latency by adding additional replicas $r > 1$ is very small as compared to the $r = 1, l = 1$ case because it has a lighter tail. In both Fig. 11 and Fig. 12 we observe that adding redundancy, that is $r \geq 1$ significantly reduces latency for a small cost, in comparison with the baseline case ($p = 0$), which is also equivalent to $r = 0, l = 1$ with any p .

5 Proofs of Single-fork Analysis

In this section we give proofs of the results in Section 3.

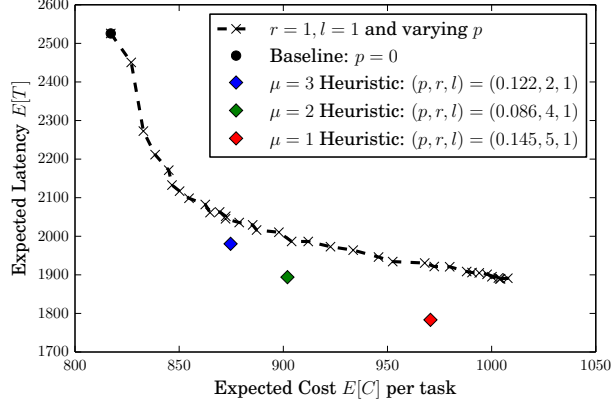


Figure 11: The latency-cost trade-off of policies found by Algorithm 2 with $\mu = 1, 2, 3$, for a Google cluster job with 1017 tasks. The $r = 1$ and $l = 1$ (parameters of back-up tasks in MapReduce) trade-off is also shown for comparison.

5.1 Latency and Cost Analysis for general F_X

Proof of Theorem 1. The expected latency $\mathbb{E}[T]$ can be divided into two parts: before and after replication.

$$\begin{aligned}
 \mathbb{E}[T] &= \mathbb{E}[T^{(1)}] + \mathbb{E}[T^{(2)}], \\
 &= \mathbb{E}[X_{(1-p)n:n}] + \mathbb{E}\left[\max_{j=1,2,\dots,pn} Y_j\right], \\
 &= F_X^{-1}(1-p) + \mathbb{E}[Y_{pn:pn}].
 \end{aligned} \tag{25}$$

The time before forking $T^{(1)}$ is the time until $(1-p)n$ of the n tasks launched at time 0 finish. Thus, its expected value $\mathbb{E}[T^{(1)}]$ is the expectation of the $(1-p)n^{th}$ order statistic $X_{(1-p)n:n}$ of n i.i.d. random variables with distribution F_X . By the Central Value Theorem stated as Theorem 10, for $n \rightarrow \infty$, this term goes to inverse CDF value $F_X^{-1}(1-p)$. At this forking point, the scheduler introduces replicas of the pn straggling tasks. The distribution F_Y of the residual execution time (minimum over the $r+1$ replicas) of each straggling task is given by Lemma 2. Thus the term $\mathbb{E}[T^{(2)}]$ in (25) is the expected value of the maximum of pn i.i.d. random variables with distribution F_Y .

Similarly, we can analyze the expected cost before and after forking. Recall from Definition 3 that the expected cost $\mathbb{E}[C]$ is the sum of the running times of all machines, normalized by the

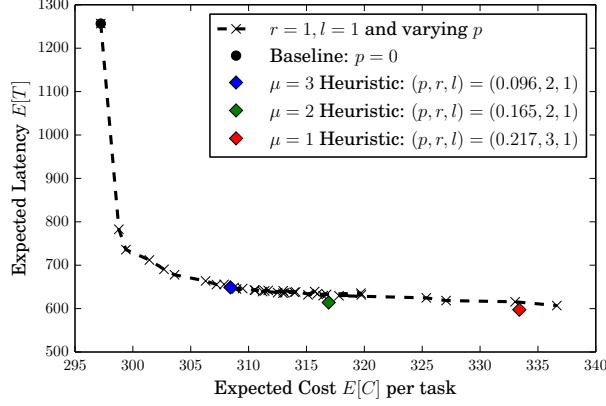


Figure 12: The latency-cost trade-off of policies found by Algorithm 2 with $\mu = 1, 2, 3$, for a Google cluster job with 488 tasks. The $r = 1$ and $l = 1$ (parameters of back-up tasks in MapReduce) trade-off is also shown for comparison.

number of tasks n .

$$\mathbb{E}[C] = \mathbb{E}[C^{(1)}] + \mathbb{E}[C^{(2)}], \quad (26)$$

$$\mathbb{E}[C^{(1)}] = \frac{1}{n} \sum_{i=1}^{(1-p)n} \mathbb{E}[X_{i:n}] + \frac{np}{n} \mathbb{E}[T^{(1)}], \quad (27)$$

$$= \frac{1}{n} \sum_{i=1}^{(1-p)n} F_X^{-1}\left(\frac{i}{n}\right) + pF_X^{-1}(1-p), \quad (28)$$

$$= \int_0^{1-p} F_X^{-1}(h)dh + pF_X^{-1}(1-p), \quad (29)$$

$$\mathbb{E}[C^{(2)}] = \frac{1}{n} \sum_{j=1}^{pn} (r+1)\mathbb{E}[Y_j], \quad (30)$$

$$= (r+1)p \cdot \mathbb{E}[Y]. \quad (31)$$

The cost before forking $\mathbb{E}[C^{(1)}]$ consists of two parts: the cost for the $(1-p)n$ tasks that finish first, and the cost for the pn straggling tasks. The first term in (27) is the sum of the expected values of the smallest $(1-p)n$ execution times. Using Theorem 10, we can show that the i^{th} term in the summation goes to $F_X^{-1}(i/n)$ as $n \rightarrow \infty$. Expressing the sum as an integral over $h = i/n$ we get the first term in (29). The second term in (27), is the normalized running time of the pn straggling tasks before forking. Substituting $\mathbb{E}[T^{(1)}]$ from (25) and simplifying, we get (29).

The cost after forking, $\mathbb{E}[C^{(2)}]$ is the normalized sum of the runtimes of the $r+1$ replicas of each of the pn straggling tasks. By Lemma 2, the residual execution time of the j^{th} straggling task is $Y_j \sim F_Y$. Since the scheduler kills all replicas as soon as one replica finishes, the expected runtime for the j^{th} straggling task is $(r+1)\mathbb{E}[Y_j]$. Thus, the cost in (30) is the sum of $(r+1)\mathbb{E}[Y_j]$ over the pn tasks, normalized by n . Since Y_j are i.i.d, we can reduce this to (31). \square

Proof of Lemma 2. First consider the case where we relaunch ($l = 0$) the original copy, and add r replicas for each of the pn straggling tasks. Thus, there are $r+1$ identical replicas of each task

after forking. The residual execution time distribution F_Y (after time $T^{(1)}$ when the replicas are added) of each task is the minimum of $r + 1$ i.i.d. random variables with distribution F_X . Hence,

$$\Pr(Y > y) = \Pr(\min(X_1, X_2, \dots, X_{r+1}) > y), \quad (32)$$

$$\bar{F}_Y(y) = \bar{F}_X(y)^{r+1} \quad \text{for } l = 0. \quad (33)$$

For the case without relaunching ($l = 1$), there is 1 original replica and r new replicas of each of the straggling tasks. Thus, the tail distribution $\bar{F}_Y(y) = 1 - F_Y(y)$ is given by

$$\Pr(Y > y) = \Pr(X_1 > y | X_1 > T^{(1)}) \cdot \Pr(\min(X_2, \dots, X_{r+1}) > y), \quad (34)$$

$$\bar{F}_Y(y) = \frac{\bar{F}_X(y + T^{(1)})}{\bar{F}_X(T^{(1)})} \bar{F}_X(y)^r. \quad (35)$$

As the number of tasks $n \rightarrow \infty$ by Theorem 10 we know that $T^{(1)} \rightarrow F_X^{-1}(1 - p)$. Hence we have,

$$\bar{F}_Y(y) = \frac{\bar{F}_X(y + F_X^{-1}(1 - p))}{p} \bar{F}_X(y)^r \quad \text{for } l = 1. \quad (36)$$

To prove Lemma 3, we characterize the expected maximum of a large number of random variables using Theorem 11. First, we state Lemma 9 which implies that the domain of attraction (see Theorem 12) of F_Y is same as that of F_X .

Lemma 9 (Domain of attraction for F_Y). *Given a single fork policy $\pi_{\text{SF}}(p, r, l; n)$ with $0 < p < 1$,*

1. *if $F_X \in \text{DA}(\Lambda)$, then $F_Y \in \text{DA}(\Lambda)$;*
2. *if $F_X \in \text{DA}(\Phi_\xi)$, then $F_Y \in \text{DA}(\Phi_{(r+1)\xi})$;*
3. *if $F_X \in \text{DA}(\Psi_\xi)$, then $F_Y \in \text{DA}(\Psi_{((1-l)r+1)\xi})$.*

The proof follows from Lemma 2 and Theorem 12 and is omitted here.

Proof of Lemma 3. We can use Lemma 9 to find the domain of attraction of F_Y . Then from (55) we have

$$\mathbb{E}[Y_{n:n}] = \tilde{a}_n \mathbb{E}[G(y)] + \tilde{b}_n,$$

where $\mathbb{E}[G(y)]$ can be found using Theorem 11 and Lemma 13. □

5.2 Latency and Cost Analysis for Pareto F_X

We prove Theorem 4, which evaluates the latency $\mathbb{E}[T]$ and computing cost $\mathbb{E}[C]$ metrics when the task execution time distribution F_X is the Pareto, as defined in (12).

Proof of Theorem 4. From Theorem 1 we have

$$\begin{aligned} \mathbb{E}[T] &= F_X^{-1}(1 - p) + \mathbb{E}[Y_{pn:pn}], \\ &= x_m p^{-1/\alpha} + \tilde{a}_{pn} \mathbb{E}[\Phi_{(r+1)\alpha}], \end{aligned} \quad (37)$$

$$= x_m p^{-1/\alpha} + \tilde{a}_{pn} \Gamma\left(1 - \frac{1}{(r+1)\alpha}\right). \quad (38)$$

$$\mathbb{E}[C] = \int_0^{1-p} F_X^{-1}(h)dh + pF_X^{-1}(1-p) + (r+1)p \cdot \mathbb{E}[Y], \quad (39)$$

$$\begin{aligned} &= x_m \int_0^{1-p} (1-h)^{-1/\alpha} dh + px_m p^{-1/\alpha} + (r+1)p \cdot \mathbb{E}[Y], \\ &= x_m \frac{\alpha}{\alpha-1} [1 - p^{1-1/\alpha}] + x_m p^{1-1/\alpha} + (r+1)p \cdot \mathbb{E}[Y], \\ &= x_m \frac{\alpha}{\alpha-1} - x_m \frac{p^{1-1/\alpha}}{\alpha-1} + (r+1)p \cdot \mathbb{E}[Y]. \end{aligned} \quad (40)$$

To obtain (37) we first observe that since F_X is Pareto, by Theorem 12 it falls into the Fréchet domain of attraction, i.e. $F_X \in \text{DA}(\Phi_\alpha)$. Then using Lemma 9 we can show that $F_Y \in \text{DA}(\Phi_{(r+1)\alpha})$. Subsequently, using Theorem 11 and Lemma 13 we get (38). To derive the expected cost (40) we substitute $F_X^{-1}(h) = x_m(1-h)^{-1/\alpha}$ in the first and second terms in (39) and simplify the expression.

To find \tilde{a}_{pn} and $\mathbb{E}[Y]$ in (38) and (40) respectively we consider the cases of relaunching ($l=0$) and no relaunching ($l=1$) separately.

Case 1: Relaunching ($l=0$)

In the single-fork policy with relaunching ($l=0$), the scheduler waits for $(1-p)n$ tasks to finish and then relaunches each of the pn straggler tasks on a new machine.

$$\begin{aligned} Y &= \min(X_1, X_2, \dots, X_{r+1}), \\ Y &\sim \text{Pareto}((r+1)\alpha, x_m). \end{aligned} \quad (41)$$

From (59) in Theorem 11 we can evaluate \tilde{a}_{pn} as follows

$$\begin{aligned} \tilde{a}_{pn} &= F_Y^{-1}\left(1 - \frac{1}{pn}\right), \\ &= x_m(pn)^{1/\alpha}. \end{aligned}$$

And $\mathbb{E}[Y]$ of (41) can be evaluated as

$$\mathbb{E}[Y] = \frac{(r+1)\alpha}{(r+1)\alpha-1} x_m. \quad (42)$$

Case 2: No Relaunching ($l=1$)

In the single-fork policy with no relaunching ($l=1$), the scheduler keeps the original copy, and adds r additional replicas for each straggling task. Using Lemma 2 we can show that

$$\bar{F}_Y(y) = \frac{1}{p} \left(\frac{x_m}{y}\right)^{\alpha r} \left(\frac{x_m}{y + x_m p^{-1/\alpha}}\right)^{\alpha}. \quad (43)$$

From (59) in Theorem 11, $\tilde{a}_{pn} = \bar{F}_Y^{-1}\left(\frac{1}{pn}\right)$, which simplifies to

$$(pn)^{1/\alpha} = \left(1 + \frac{\tilde{a}_{pn}}{x_m p^{-1/\alpha}}\right) \left(\frac{\tilde{a}_{pn}}{x_m}\right)^r,$$

which simplifies to (17).

The expected value of Y can be found by numerically integrating $\bar{F}_Y(y)$ in (43) over its support. \square

Proof of Lemma 5. For given r , the expected latency is lower with relaunching when p satisfies,

$$\begin{aligned}\mathbb{E}[T]^{(l=1)} &\geq \mathbb{E}[T]^{(l=0)}, \\ \tilde{a}_{pn}^{(l=1)} &\geq \tilde{a}_{pn}^{(l=0)}, \\ n^{1/\alpha} x_m^{r+1} &\geq x_m p^{-1/\alpha} (pn)^{\frac{r}{(r+1)\alpha}} x_m^r + (pn)^{\frac{1}{\alpha}} x_m^{r+1}, \\ 1 &\geq (pn)^{-1/(r+1)\alpha} + p^{1/\alpha}.\end{aligned}\quad \square$$

Proof of Lemma 6. Given r and relaunching ($l = 0$), the single-fork policy is sub-optimal in both $\mathbb{E}[T]$ and $\mathbb{E}[C]$ if p satisfies

$$\frac{d\mathbb{E}[T]}{dp} \cdot \frac{d\mathbb{E}[C]}{dp} > 0.$$

Substituting $\mathbb{E}[T]$ and $\mathbb{E}[C]$ from Theorem 4 and simplifying, we get (20). \square

5.3 Latency and Cost Analysis for Shifted Exponential F_X

Now we prove Theorem 7, which gives the latency-cost trade-off when the distribution of the execution time X is a shifted exponential given by (21).

Proof of Theorem 7.

$$\begin{aligned}\mathbb{E}[T] &= F_X^{-1}(1-p) + \mathbb{E}[Y_{pn:pn}], \\ &= \Delta - \frac{1}{\lambda} \ln p + \tilde{a}_{pn} \mathbb{E}[\Lambda] + \tilde{b}_{pn},\end{aligned}\quad (44)$$

$$= \Delta - \frac{1}{\lambda} \ln p + \tilde{a}_{pn} \gamma_{\text{EM}} + \tilde{b}_{pn}.\quad (45)$$

$$\mathbb{E}[C] = \int_0^{1-p} F_X^{-1}(h) dh + p F_X^{-1}(1-p) + (r+1)p \cdot \mathbb{E}[Y],\quad (46)$$

$$\begin{aligned}&= \int_0^{1-p} \left(\Delta - \frac{1}{\lambda} \ln(1-h) \right) dh + p \left(\Delta - \frac{1}{\lambda} \ln p \right), \\ &\quad + (r+1)p \cdot \mathbb{E}[Y],\end{aligned}\quad (47)$$

$$\begin{aligned}&= \Delta + \frac{1}{\lambda} (p \ln p + (1-p)) + p \Delta - \frac{p}{\lambda} \ln p, \\ &\quad + (r+1)p \cdot \mathbb{E}[Y],\end{aligned}\quad (48)$$

$$= \Delta(1+p) + \frac{1-p}{\lambda} + (r+1)p \cdot \mathbb{E}[Y].\quad (49)$$

To find $\mathbb{E}[Y]$, \tilde{a}_{pn} and \tilde{b}_{pn} we consider the cases of relaunching ($l = 0$) and no relaunching ($l = 1$) separately.

Case 1: Relaunching ($l = 0$)

$$Y = \min \{X_1, X_2, \dots, X_{r+1}\} \quad (50)$$

$$\sim \text{SExp}(\Delta, (r+1)\lambda) \quad (51)$$

$$\mathbb{E}[Y] = \Delta + \frac{1}{(r+1)\lambda} \quad (52)$$

Based on Theorem 12, for $\eta(y) = 1/((r+1)\lambda)$ we have

$$\lim_{y \rightarrow \omega(F_Y)} \frac{\bar{F}_Y(y + u\eta(y))}{\bar{F}_Y(y)} = e^{-u}. \quad (53)$$

By Theorem 11 and Theorem 12, the maximum of shifted exponential belongs to the Gumbel family with

$$\begin{aligned} \tilde{a}_{pn} &= \frac{1}{\lambda(1+r)}, \\ \tilde{b}_{pn} &= \bar{F}_Y^{-1}(1/n) = \Delta + \frac{\ln(pn)}{\lambda(r+1)}. \end{aligned}$$

Case 2: No Relaunching ($l = 1$)

In the case of no relaunching,

$$Y = \min \{ \text{Exp}(\lambda), \Delta + \text{Exp}(r\lambda) \}.$$

Note that the first term does not include Δ because for large n the original task would have run for at least Δ seconds. Thus the tail distribution of Y is given by

$$\bar{F}_Y(y) = \begin{cases} e^{-\lambda y} & 0 < y < \Delta, \\ e^{\lambda r \Delta} e^{-\lambda(r+1)y} & y \geq \Delta. \end{cases} \quad (54)$$

The expected value $\mathbb{E}[Y]$ is the integration of $\bar{F}_Y(y)$ over its support.

$$\begin{aligned} \mathbb{E}[Y] &= \int_0^\Delta e^{-\lambda y} dy + \int_\Delta^\infty e^{\lambda r \Delta} e^{-\lambda(r+1)y} dy, \\ &= \frac{1 - e^{-\lambda \Delta}}{\lambda} + \frac{e^{-\lambda \Delta}}{\lambda(r+1)}. \end{aligned}$$

By Theorem 11 and Theorem 12 similar to the relaunching case we have

$$\begin{aligned} \tilde{a}_{pn} &= 1/[\lambda(1+r)], \\ \tilde{b}_{pn} &= \bar{F}_Y^{-1}(1/n) = \frac{r}{r+1} \Delta + \frac{\ln(pn)}{\lambda(r+1)}. \end{aligned} \quad \square$$

Proof of Lemma 8. For the shifted exponential distribution it is clear that no relaunching always gives a lower latency. We now find conditions on λ and Δ for which no relaunching gives lower cost. Define $\beta = \lambda\Delta$, then

$$\begin{aligned} &\lambda \mathbb{E}[C(l=1)] > \lambda \mathbb{E}[C(l=0)], \\ \Leftrightarrow &n + pn \left[\lambda \Delta + r(1 - e^{-\lambda \Delta}) \right] > n + pn \lambda(r+2)\Delta, \\ \Leftrightarrow &\beta r + \beta - r + r e^{-\beta} < 0. \end{aligned}$$

And note that the function $g(\beta) = \beta r + \beta - r + r e^{-\beta}$ is monotonically increasing since $g'(\beta) > 0$ for any $r \in \mathbb{Z}^+$. \square

6 Concluding Remarks

6.1 Major Implications

In this paper we show that replication of the slowest tasks in a job (the stragglers) is a powerful way to reduce latency in large-scale parallel computing. We characterize the trade-off between latency and computing cost for a set of replication strategies called single-fork policies. The policy used in practical systems such as MapReduce belongs to this set of single-fork policies. A non-intuitive insight we get from this analysis is that in certain scenarios it is possible to reduce latency and cost simultaneously. We also propose a heuristic algorithm to find the scheduling policy which achieves a good trade-off between latency and cost. Experiments on Google Traces data show that policies found by the heuristic algorithm can give a better latency-cost trade-off than the back-up tasks option used in MapReduce.

6.2 Future Perspectives

Although we focus on single-fork policies in this work, the analysis can be extended to multi-fork policies where the scheduler can add or kill replicas at multiple instances during the execution. We also plan to consider queueing of tasks at machines and analyze the percentile latency in addition to the expected value considered here. Another future direction is to analyze the performance of approximate computing where only a subset of the tasks of a job are required to be completed, for example information retrieval and machine learning applications.

In this work, we assume prior knowledge of the execution time distribution F_X when designing the task replication strategy. An interesting research direction is to develop an adaptive scheduling algorithm that simultaneously estimates the distribution and schedules task replication. This shares some similarity to the celebrated multi-arm bandit problem, with an exploration-exploitation trade-off between estimating the distribution F_X , and using the current estimate to design a task replication policy.

Our analytical framework can be used in other applications where the response time of the components is stochastic. For example, in crowdsourcing, each worker can take a variable amount of time to complete a task. Then the overall latency can be modeled in the same way as our work, with the cost function being equal to the number of workers [20].

References

- [1] Google cluster data. <http://code.google.com/p/googleclusterdata/>.
- [2] G. Ananthanarayanan, A. Ghodsi, and I. Stoica S. Shenker. Effective straggler mitigation: Attack of the clones. In *USENIX Conference on Networked Systems Design and Implementation*, pages 185–198, 2013.
- [3] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris. Reining in the Outliers in Map-Reduce Clusters Using Mantri. In *USENIX Conference on Operating Systems Design and Implementation*, pages 1–16, 2010.
- [4] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

- [5] W. Cirne, F. Brasileiro, D. Paranhos, L. Fabrício W. Góes, and W. Voorsluys. On the efficacy, efficiency and emergent behavior of task replication in large distributed systems. *Parallel Computing*, 33(3):213–234, 2007.
- [6] H. A. David and H. N. Nagaraja. *Order statistics*. John Wiley, Hoboken, N.J., 2003.
- [7] L. de Haan and A. Ferreira. *Extreme value theory an introduction*. Springer, New York, 2006.
- [8] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. 51(1):107–113, 2008.
- [9] B. Efron and R. Tibshirani. Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical science*, pages 54–75, 1986.
- [10] G. Ghare and S. T. Leutenegger. Improving speedup and response times by replicating parallel programs on a SNOW. In *International conference on Job Scheduling Strategies for Parallel Processing*, pages 264–287, January 2005.
- [11] J. Dean and L. Barroso. The Tail at Scale. *Communications of the ACM*, 56(2):74–80, 2013.
- [12] G. Joshi, Y. Liu, and E. Soljanin. Coding for fast content download. In *Allerton Conference on Communication, Control and Computing*, pages 326–333, October 2012.
- [13] G. Joshi, Y. Liu, and E. Soljanin. On the Delay-Storage Trade-off in Content Download from Coded Distributed Storage Systems. *IEEE JSAC*, pages 989 – 997, May 2014.
- [14] W. Neiswanger, C. Wang, and E. Xing. Asymptotically exact, embarrassingly parallel MCMC. *arXiv:1311.4780 [cs, stat]*, November 2013.
- [15] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica. Sparrow: Distributed, low latency scheduling. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 69–84, 2013.
- [16] C. Reiss, A. Tumanov, G. Ganger, R. H. Katz, and M. A. Kozuch. Towards understanding heterogeneous clouds at scale: Google trace analysis. *Intel Science and Technology Center for Cloud Computing, Tech. Rep*, 2012.
- [17] N. B. Shah, K. Lee, and K. Ramchandran. When do redundant requests reduce latency? In *IEEE International Symposium on Information Theory (ISIT)*, 2014.
- [18] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop Distributed File System. In *IEEE Symposium of Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010.
- [19] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker. Low latency via redundancy. *arXiv:1306.3707 [cs]*, June 2013.
- [20] D. Wang. *Computing with Unreliable Resources: Design, Analysis and Algorithms*. PhD thesis, Massachusetts Institute of Technology, 2014.
- [21] D. Wang, G. Joshi, and G. Wornell. Efficient task replication for fast response times in parallel computation. *ACM Sigmetrics short paper*, June 2014.
- [22] Wikipedia. Embarrassingly parallel — Wikipedia, the free encyclopedia, February 2013.

- [23] M. Zaharia, A. Konwinski, A. Joseph, R. Katz, and I. Stoica. Improving MapReduce performance in heterogeneous environments. In *USENIX Conference on Operating Systems Design and Implementation*, pages 29–42, 2008.

In this section we introduce some definitions and results in order statistics that play an important role in the analysis of scheduling policies in Section 3. In particular, we describe how order statistics in different regimes have different concentration behavior in Appendices A and B.

A Central order statistics

For an order statistic $X_{k:n}$, we called it a central order statistic if $k \approx np$ for some $p \in (0, 1)$. In this case, $X_{k:n}$ is asymptotically normal, concentrated around the p -th quantile of X , as indicated by the following result called the Central Value Theorem (Theorem 10.3 in [6]).

Theorem 10 (Central Value Theorem). *Given $X_1, X_2, \dots, X_n \stackrel{i.i.d.}{\sim} F$, if $0 < p < 1$ and $0 < f(x_p) < \infty$, where $x_p = F^{-1}(p)$, then for $k = k(n)$ such that $k = np + o(\sqrt{n})$,*

$$X_{k:n} \xrightarrow{P} N\left(x_p, \frac{p(1-p)}{nf^2(x_p)}\right)$$

where $f(\cdot)$ is the p.d.f. corresponds to F and \xrightarrow{P} denotes convergence in probability as $n \rightarrow \infty$.

B Extreme order statistics

Extreme value theory (EVT) is an asymptotic theory of extremes, i.e., minima and maxima. It shows that if a distribution belongs to one of three families of distributions Theorem 12), then its maxima can be well characterized asymptotically as given by Theorem 11, which is also referred to as the Fisher-Tippett-Gnedenko Theorem (Theorem 1.1.3 in [7]).

Theorem 11 (Extreme Value Theorem). *Given $X_1, \dots, X_n \stackrel{i.i.d.}{\sim} F$, if there exist sequences of constants $a_n > 0$ and $b_n \in \mathbb{R}$ such that*

$$\mathbb{P}[(X_{n:n} - b_n)/a_n \leq x] \rightarrow G(x) \quad (55)$$

as $n \rightarrow \infty$ and $G(\cdot)$ is a non-degenerate distribution. The extreme value distribution $G(x)$ and the values of a_n and b_n depend on the domain of attraction (and hence the tail behavior) of F_X given by Theorem 12.

1. For $F_X \in \text{DA}(\Lambda)$,

$$a_n = \eta(F^{-1}(1 - 1/n)), \quad (56)$$

$$b_n = F^{-1}(1 - 1/n) \quad (57)$$

$$G(x) = \Lambda(x) = \exp\{-\exp(-x)\} \quad (58)$$

where $\Lambda(x)$ is called the Gumbel distribution.

2. For $F_X \in \text{DA}(\Phi_\xi)$,

$$a_n = F^{-1}(1 - 1/n), \quad (59)$$

$$b_n = 0, \quad (60)$$

$$G(x) = \Phi_\xi(x) = \begin{cases} 0 & x \leq 0 \\ \exp\{-x^{-\xi}\} & x > 0 \end{cases}. \quad (61)$$

where $\Phi_\xi(x)$ is called the Fréchet distribution.

3. For $F_X \in \text{DA}(\Psi_\xi)$,

$$a_n = \omega(F) - F^{-1}(1 - 1/n), \quad (62)$$

$$b_n = \omega(F), \quad (63)$$

$$G(x) = \Psi_\xi(x) = \begin{cases} \exp\left\{-(-x)^\xi\right\} & x < 0, \\ 1 & x \geq 0. \end{cases} \quad (64)$$

where $\Psi_\xi(x)$ is called the reversed-Weibull distribution.

Theorem 12 (Domains of attraction). A distribution function F_X has one of the following domains of attraction if it satisfies the conditions of the extreme value distribution $G(x)$ if and only if

1. $F_X \in \text{DA}(\Lambda)$ if and only if there exists $\eta(x) > 0$ such that

$$\lim_{x \rightarrow \omega(F)^-} \frac{\bar{F}(x + t\eta(x))}{\bar{F}(x)} = e^{-t};$$

2. $F_X \in \text{DA}(\Phi_\xi)$ if and only if $\omega(F) = \infty$ and

$$\lim_{x \rightarrow \infty} \frac{\bar{F}(tx)}{\bar{F}(x)} = t^{-\xi}, \quad t > 0;$$

3. $F_X \in \text{DA}(\Psi_\xi)$ if and only if $\omega(F) < \infty$ and

$$\lim_{x \rightarrow 0^+} \frac{\bar{F}(\omega(F) - tx)}{\bar{F}(\omega(F) - x)} = t^\xi, \quad t > 0;$$

where $\omega(x) = \sup\{x : F_X(x) < 1\}$, the upper end point of the distribution F_X .

Intuitively, $F \in \text{DA}(\Lambda)$ corresponds to the case that \bar{F} has an exponentially decaying tail, $F \in \text{DA}(\Phi_\xi)$ corresponds to the case that \bar{F} has heavy tail (such as polynomially decaying), and $F \in \text{DA}(\Psi_\xi)$ corresponds to the case that \bar{F} has a short tail with finite upper bound.

Lemma 13 (Expected Extreme Values).

$$\mathbb{E}[\Lambda] = \gamma_{\text{EM}},$$

$$\mathbb{E}[\Phi_\xi] = \begin{cases} \Gamma(1 - 1/\xi) & \xi > 1 \\ +\infty & \text{otherwise,} \end{cases}$$

$$\mathbb{E}[\Psi_\xi] = -\Gamma(1 + 1/\xi),$$

where γ_{EM} is the Euler-Mascheroni constant and $\Gamma(\cdot)$ is the Gamma function, i.e.,

$$\Gamma(t) \triangleq \int_0^\infty x^{t-1} e^{-x} dx.$$

We can also characterize the limit distribution of the sample extreme $X_{1:n}$ analogously via Theorem 12 by

$$X_{1:n} = \min\{X_1, \dots, X_n\} = -\max\{-X_1, \dots, -X_n\}.$$

It is worth noting that the distribution function for $-X$ may be in a different domain of attraction from that of X .